Chapter 1: What the Heck Is VBA?

In This Chapter

- **Understanding Visual Basic for Applications (VBA)**
- **Using the Visual Basic Editor**
- **Discovering code as you go**

Visual Basic for Applications — often abbreviated *VBA* — is a programming language you can use to extend the functionality of Microsoft Access and other products in the Microsoft Office suite of programs. A *programming language* is a means of writing instructions for the computer to *execute* (perform). Programmers often refer to the written instructions as *code* because the instructions aren't in plain English. Rather, they're in a code that the computer can interpret and execute.

You can create sophisticated Access databases without using VBA at all. In most cases, the other objects offered by Access — tables, queries, forms, reports, and macros — offer more than enough flexibility and power to create just about any database imaginable. But once in a while, you come across a situation where you want to do something that none of those other objects can do. That's where VBA comes in. If you can find no other way to accomplish some goal in Access, writing code is usually the solution.

COPYRIGHTED MATERIAL

Finding VBA Code

So what the heck is VBA code, anyway? To the untrained eye, VBA code looks like gibberish — perhaps some secret code written by aliens from another planet. But to Access, the code represents very specific instructions on how to perform some task.

Within any given database, Access stores code in two places:

- **Class modules (Code-Behind Forms):** Every form and report you create automatically contains a *class module* (also called a *code-behind form*), as illustrated in Figure 1-1. The class module for a given form or report is empty unless you place controls that require VBA code on that form or report.

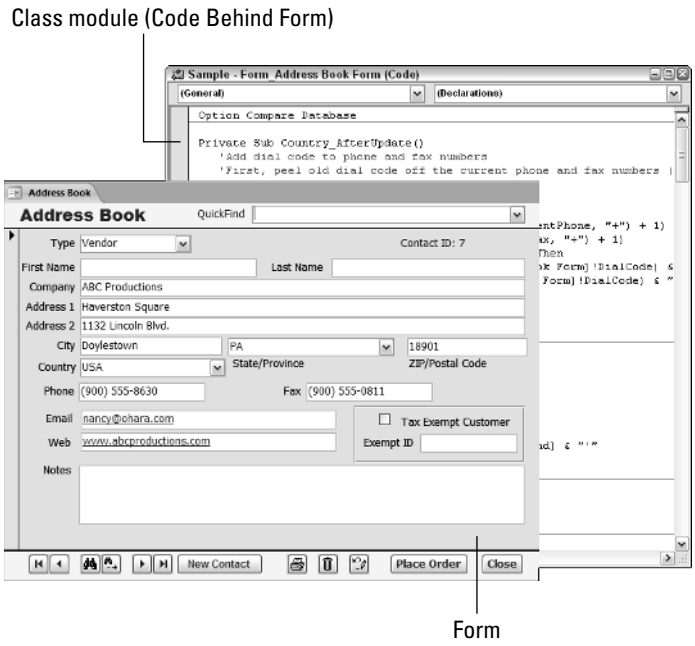

Figure 1-1: Every form and report has a class module behind it.

- **Standard modules:** Code can also be stored in *standard modules.* Code in standard modules is accessible to all objects in your database, not just a single form or report.

Opening a class module

If you want to view or change the code for a form or report's class module, first open, in Design view, the form or report to which the module is attached. Then click the View Code button, shown near the mouse pointer in Figure 1-2.

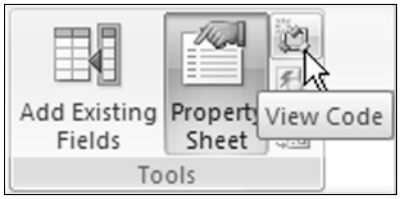

Figure 1-2: The View Code button.

You can also get to a class module from the Event tab of the Property sheet in the Design View window. The Property sheet allows you to zoom right in on the VBA code that's associated with a given control. For example, some controls contain code created by wizards. When you click such a control and then click the Events tab in the Property sheet, the property value chose `[Event Procedure]`. When you click `[Event Procedure]`, you see a button with three dots, like the one near the mouse pointer in Figure 1-3. That's the Build button. Click it to see the code that executes in response to the event.

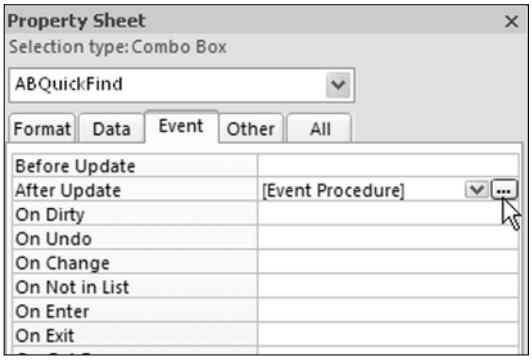

Figure 1-3: Look for the code that executes in response to the event.

To write custom code for a control, select the control in Design view, open the Property sheet, click the Event tab, click the event to which you want to attach some custom code, click the Build button, and then choose Code Builder.

After you open a module, you're taken to an entirely separate program window called the *Visual Basic Editor,* where you see the module in all its glory.

Creating or opening a standard module

Standard modules contain VBA code that isn't associated with a specific form or report. The code in a standard module is available to all tables, queries, forms, reports, macros, and other modules in your database. You won't see Module as an option when you're viewing All Access Objects in the shutter bar until you create at least one standard module. You have to go looking for options to create and work with modules.

To create a new module, click the Create tab. Then click the arrow under the Macros button and choose Module (Figure 1-4). The Visual Basic Editor opens.

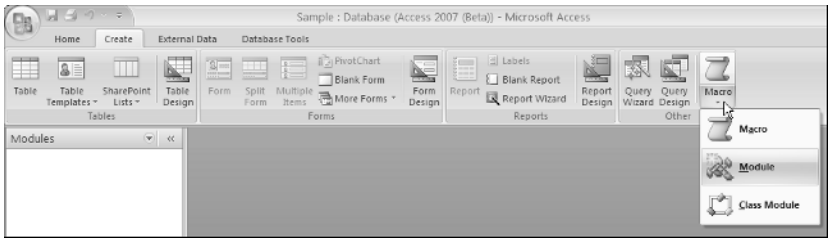

Figure 1-4: Create a new module.

Standard modules don't show up automatically in the shutter bar, not even when you're viewing all Access object types. To view standard modules in your database, you have to click the drop-down button and choose Modules, as in Figure 1-5. If you've already created and saved a standard module, you can open it by double-clicking its name. If the current database contains no standard Modules, you won't even see Modules as a category.

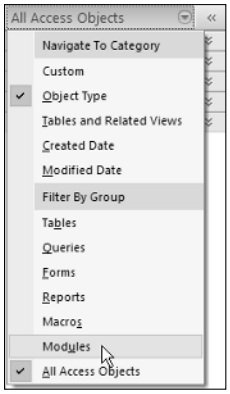

Figure 1-5: Open a pane to see standard modules.

Regardless of whether you create or open a module, you end up in the Visual Basic Editor. The editor is a completely separate program with its own taskbar button. The editor retains the old-style Windows look and feel. We cover that in more detail in a moment. For now, keep in mind that you can close the Visual Basic Editor and return to Access at any time. Just click the Close (X) button in the Editor's upper-right corner.

Enabling VBA Code

Like any programming language, people can use VBA to create code that does good things or code that does bad things. Whenever you open a database that contains code, Access displays a warning in the Security bar. The warning doesn't mean that there's "bad code" in the database; it just means that there *is* code in the database. Access has no way of determining whether the code is beneficial or malicious. That's a judgment call only a human can make.

If you trust the source of that code, you have to click the Enable Content button to make the code executable. Otherwise, the code is disabled, as are many features of the Visual Basic Editor.

How code is organized

All modules organize their code into a `Declaration` section at the top, followed by individual procedures, as shown in Figure 1-6. The `Declaration` section contains options, written in code format, that apply to all procedures in the module. Each procedure is also a chunk of VBA code that, when executed, performs a specific set of steps.

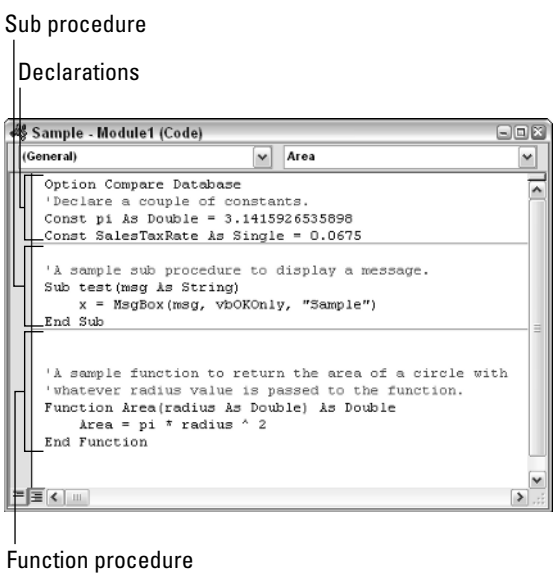
Sub procedure
Declarations
Sample - Module1 (Code)
(General)
Area
Option Compare Database
'Declare a couple of constants.
Const pi As Double = 3.1415926535898
Const SalesTaxRate As Single = 0.0675
'A sample sub procedure to display a message.
Sub test(msg As String)
x = MsgBox(msg, vbOKOnly, "Sample")
End Sub
'A sample function to return the area of a circle with
'whatever radius value is passed to the function.
Function Area(radius As Double) As Double
Area = pi * radius ^ 2
End Function
Function procedure

Figure 1-6: Modules consist of declarations and procedures.

Procedures in a module fall into two major categories: sub procedures and function procedures. Both types of procedures use VBA code to perform some task. The next sections outline some subtle differences in how and where they're used.

Sub procedures

A *sub procedure* is one or more lines of code that make Access perform a particular task. Every sub procedure starts with the word `Sub` (or `Private Sub`) and ends with `End Sub`, using one of the following general structures:

```
Sub name()
   ...code...
End Sub

Private Sub name()
   ...code...
End Sub
```

name is the name of the procedure, and `...`*code*`...` is any amount of VBA code.

Text that appears to be written in plain English within a module represents *programmer comments* — notes for other programmers. The computer ignores the comments. Every comment starts with an apostrophe (`'`).

Function procedures

A *function procedure* is enclosed in `Function...End Function` statements, as the following code shows:

```
Function name()
   <...code...>
End Function
```

Unlike a sub procedure, which simply performs some task, a function procedure performs a task and returns a value. In fact, an Access function procedure is no different from any of the built-in functions you use in Access expressions. And you can use a custom function procedure wherever you can use a built-in procedure.

Using the Visual Basic Editor

Regardless of how you open a module, you end up in the Visual Basic Editor. The Visual Basic Editor is where you write, edit, and test your VBA code. The Visual Basic Editor is entirely separate from the Access program window. If

you click outside the Visual Basic Editor window, the window may disappear as whatever window you clicked comes to the front.

The Visual Basic Editor retains the view it had in previous versions of Access. There is no Ribbon or shutter bar. In fact, the Visual Basic Editor is virtually identical to Microsoft's *Visual Studio,* the IDE (Integrated Development Environment) used for all kinds of programming with Microsoft products.

Like all program windows, the Visual Basic Editor has its own Windows taskbar button, as shown in the top half of Figure 1-7. If the taskbar is particularly crowded with buttons, the editor and Access may share a taskbar button, as in the bottom half of Figure 1-7. If you suddenly lose the VBA Editor window, click its taskbar button to bring the window back to the top of the stack of program windows on your desktop.

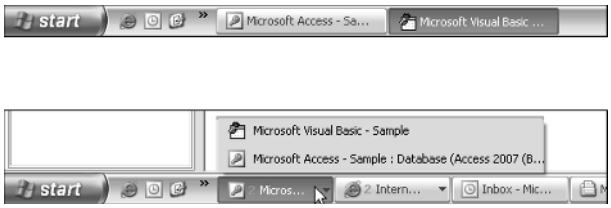

Figure 1-7: Taskbar buttons for Access and the Visual Basic Editor.

In most versions of Windows, you can right-click the Windows taskbar and choose the Tile Windows Vertically option from the shortcut menu to make all open program windows visible on-screen without overlap.

Talkin' the talk

Programmers have their own slang terms to describe what they do. For example, the term *code,* which refers to the actual instructions written in a programming language, is always singular, like the terms *hardware* and *software.* You don't add *hardwares* and *softwares* to your computer system. You add hardware and software. Likewise, you never write, or cut and paste code*s.* You write, or cut and paste, code.

The term GUI (pronounced *goo*-ey) refers to Graphical User Interface. Anything you can accomplish by using a mouse (that is, without writing code) is considered part of the GUI. You create tables, queries, forms, reports, data access pages, and macros using the GUI. You only need to write code in modules.

A database may be referred to as an *app,* which is short for *application.* If a programmer says, "I created most of the app with the GUI; I hardly wrote any code at all," he means he spent most of his time creating tables, queries, forms, reports, data access pages, and macros — using the mouse — and relatively little time typing code in VBA.

The Visual Basic Editor provides many tools designed to help you write code. Most of the tools are optional and can be turned on or off using the View menu in the Visual Basic Editor menu. The windows are shown in Figure 1-8. We provide more information on each of the optional windows when they become relevant to the type of code we're demonstrating. For now, knowing how to make them appear and disappear is sufficient.

You can move and size most of the windows in the Visual Basic Editor using standard methods. For instance, you can move most windows by dragging their title bars. You size windows by dragging any corner or edge. Most of the time, you won't need to have all those optional windows open to write code. Feel free to close any optional window open in your editor by clicking its Close (X) button. To open a window, choose View from the menu, and click the name of the window you want to open.

If you have multiple monitors connected to your computer, you can put the Access window on one monitor and the Visual Basic Editor window on the other.

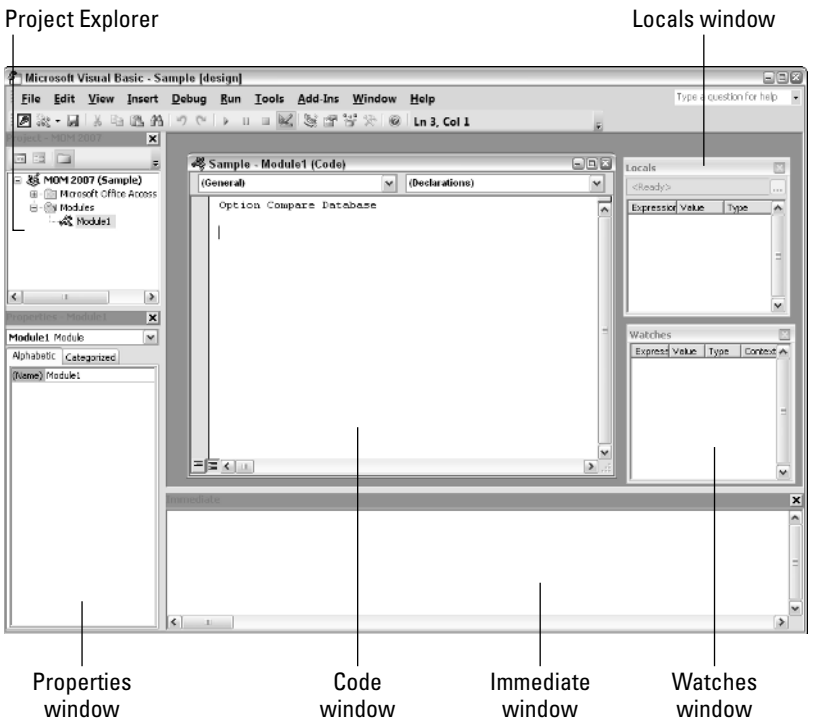

Figure 1-8: Visual Basic Editor components.

Using the Code window

The Code window is where you type your VBA code. Similar to a word processor or text editor, the Code window supports all the standard Windows text-editing techniques. You can type text and use the Backspace and Delete keys on your keyboard to delete text. You can use the Tab key to indent text. You can select text by dragging the mouse pointer through it. You can copy and paste text to, and from, the Code window. In short, the Code window is a text editor.

The Code window acts like the document window in most other programs. Click its Maximize button, shown near the mouse pointer at the top of Figure 1-9, to enlarge it. To restore it to its previous size, click the Restore Window button, shown at the bottom of that same figure.

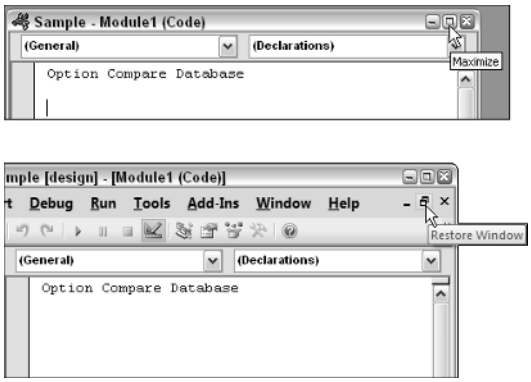

Figure 1-9: Code Windows Maximize and Restore Window buttons.

Tools in the Code window are pointed out in Figure 1-10 and summarized in the following list:

- **Object box:** When you're viewing a class module, this box shows the name of the object associated with the current code and allows you to choose a different object. In a standard module, only the word `General` appears because a standard module isn't associated with any specific form or report.
- **Procedure/Events box:** When you're viewing a class module, this box lists events supported by the object whose name appears in the Object box. When viewing a standard module, the Procedure/Events box lists the names of all procedures in that module. To jump to a procedure or event, just choose its name from the drop-down list.

- **Split bar:** This divvies up the screen for you. Drag the Split bar down to separate the Code window into two independently scrollable panes. Drag the Split bar back to the top of the scroll bar to unsplit the window.
- **Procedure view:** When clicked, it hides declarations, and only procedures are visible.
- **Full Module view:** When clicked, it makes declarations and procedures visible.
- **Sizing handle:** Drag it to size the window. (You can drag any corner or edge as well.)

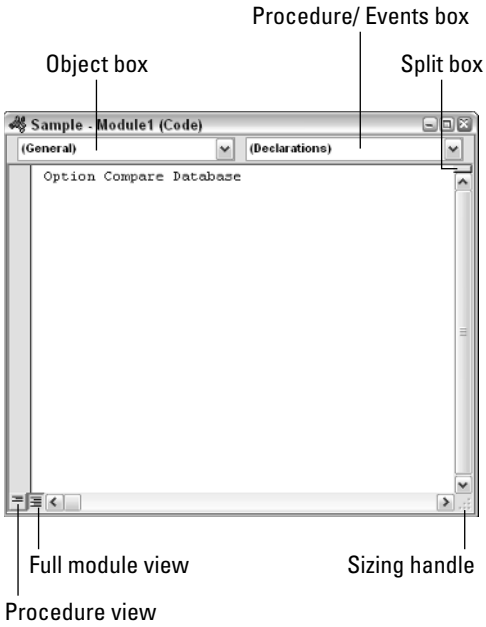

Figure 1-10: The Code window tools.

Using the Immediate window

The *Immediate window,* or *debug window,* in the Visual Basic Editor allows you to run code at any time, right on the spot. Use the Immediate window for testing and debugging (removing errors from) code. If the Immediate window isn't open in the Visual Basic Editor, you can bring it out of hiding at any time by choosing View➪Immediate Window from the editor's menu.

When the Immediate window is open, you can anchor it to the bottom of the Visual Basic Editor just by dragging its title bar to the bottom of the window. Optionally, you can make the Immediate window free-floating by dragging its title bar up and away from the bottom of the Visual Basic Editor program window. You can also dock and undock the Immediate window by right-clicking within the Immediate window and choosing the Dockable option from the shortcut menu that appears.

The Immediate window allows you to test expressions, run VBA procedures you create, and more. To test an expression, you can use the `debug.print` command, or the abbreviated `?` version, followed by a blank space and the expression. Which command you use doesn't matter, although obviously, typing the question mark is easier. You may think of the `?` character in the Immediate window as standing for *"What is . . . ?"* Typing `? 1+1` into the Immediate window and pressing Enter is like asking, "What is one plus one?" The Immediate window returns the answer to your question, `2`, as shown in Figure 1-11.

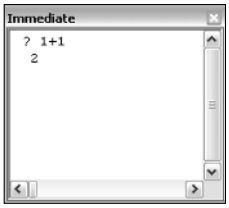

Figure 1-11: The free-floating Immediate window solves 1 + 1 calculation.

If you see a message about macro content being blocked, switch over to the Access program window and click the Enable Content button on the Security bar.

If you want to re-execute a line that you already typed into the Immediate window, you don't need to type that same line in again. Instead, just move the cursor to the end of the line that you want to re-execute and press Enter. To erase text from the Immediate window, drag the mouse pointer through whatever text you want to erase. Then press the Delete (Del) key or right-click the selected text and choose the Cut option from the shortcut menu.

You see many examples of using the Immediate window in the forthcoming chapters of this book. For the purposes of this chapter, knowing the Immediate window exists and basically how it works is enough.

Do bear in mind that the Immediate window is just for testing and debugging. The Code window is where you type (or paste in) VBA code.

Using the Object Browser

VBA code can manipulate Access objects programmatically. Remember, everything in Access is an object — tables, forms, reports, and even a single control on a form or report are objects. Every Access object you see on-screen in Access is managed either interactively or programmatically. When you work with objects in the Access program window, using your mouse and keyboard, you use Access interactively. You do something with your mouse and keyboard and the object responds accordingly.

When you write code, you write instructions that tell Access to manipulate an object *programmatically,* without user intervention. You write instructions to automate some task that you may otherwise do interactively with mouse and keyboard. In order to manipulate an object programmatically, you write code that refers to the object by name.

All the objects that make up Access and the current database are organized into an *object model,* which comprises one or more object libraries. An *object library* is an actual file on your hard drive that provides the names of objects that VBA refers to and manipulates.

Each object consists of *classes*, where each class is a single programmable object. Each class has *members,* and some members are *properties*. Properties are characteristics of the class, such as its name, or the number of items it contains. Other members are *methods,* which expose things you can do to the class programmatically.

The object model is huge and contains many libraries and classes. There's no way to memorize everything in the object model. It's just too darn big. The Visual Basic Editor provides an Object Browser that acts as a central resource for finding things as well as getting help with things in the model. It's especially useful for deciphering other peoples' code, like the examples you'll see in this book.

To view the objects that VBA can access, follow these steps to open the Object Browser:

1. **Make sure you're in the Visual Basic Editor.**

2. **Click the Object Browser button in the toolbar, choose View➪Object Browser from the menu, or press the F2 key.**

The Object Browser opens. Figure 1-12 shows the Object Browser and points out some of the major features of its window. The following list describes each component:

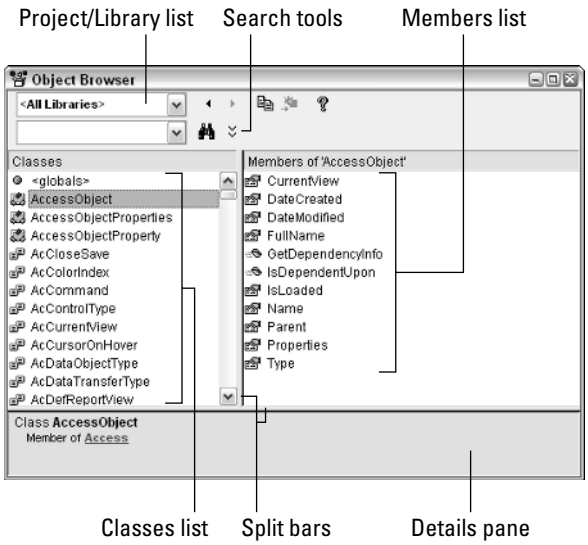

Figure 1-12: The Object Browser.

- **Project/Library list:** This allows you to choose a single library or project to work with, or `<All Libraries>`.
- **Search tools:** Use these tools to help you find information in the libraries.
- **Classes list:** This shows the names of all classes in the currently selected library or project name (or all libraries).
- **Members list:** When you click a name in the Classes list, this pane shows the members (properties, methods, events, functions, objects) that belong to that class.
- **Details pane:** When you click a member name in the Members list, the Details pane shows the syntax for using the name as well as the name of the library to which the member belongs. You can copy text from the Details pane to the Code window.
- **Split bar:** Drag the Split bar left or right to adjust the size of the panes. (Drag any edge or corner of the Object Browser window to size the window as a whole.)

Searching the Object Library

For a beginning programmer, the sheer quantity of items in the Object Browser is daunting. However, learning about the pre-written code you pick up elsewhere is useful. Suppose you find and use a procedure that has a `DoCmd` object in it. You're wondering what this `DoCmd` thingy is.

You can search the Object Library for information about any object, including `DoCmd`, by following these steps:

1. **In the Object Browser, type the word you're searching for in the Search box.**

   In this example, type **DoCmd**, as shown in Figure 1-13.

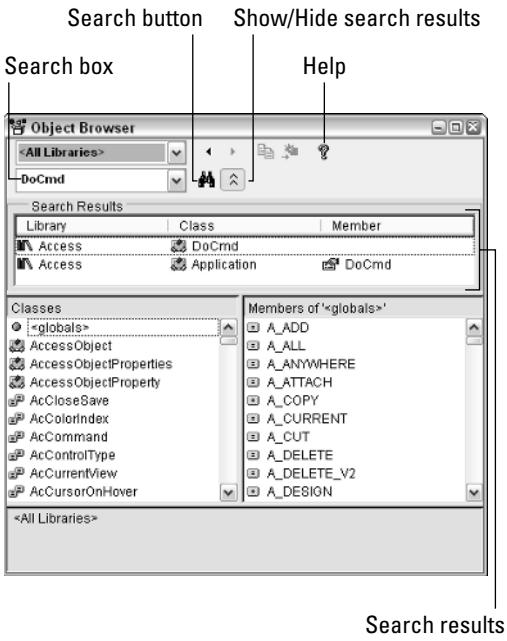

Figure 1-13: Object Browser search tools.

2. **Click the Search button.**

   The search results appear in the Search Results pane.

3. **Click the word you searched for.**

4. **Click the Help (question mark) button on the Object Browser toolbar.**

Figure 1-14 shows the Help window for the `DoCmd` object. For the absolute beginner, even the information in the Help text may be a bit advanced. However, as you gain experience and dig a little deeper into VBA, you'll find the Object Browser and Help windows useful for constructing references to objects, properties, and methods from within your code.

Access Help
Search
Access Developer Home > Access Object Model Reference > DoCmd Object
DoCmd Object
Access Developer Reference
DoCmd Object
Show All
You can use the methods of the DoCmd object to run Microsoft Access actions from Visual Basic. An action performs tasks such as closing windows, opening forms, and setting the value of controls.
Remarks
For example, you can use the OpenForm method of the DoCmd object to open a form, or use the Hourglass method to change the mouse pointer to an hourglass icon.
Most of the methods of the DoCmd object have arguments — some are required, while others are optional. If you omit optional arguments, the arguments assume the default values for the particular method. For example, the OpenForm method uses seven arguments, but only the first argument, FormName, is required. The following example shows how you can open the Employees form in the current database. Only employees with the title Sales Representative are included.
DoCmd.OpenForm "Employees", , ,"[Title] = 'Sales Representative'"
The DoCmd object doesn't support methods corresponding to the following actions:
AddMenu.
MsgBox. Use the MsgBox function.
RunApp. Use the Shell function to run another application.
Developer Reference
Connected to Office Online

Figure 1-14: Help for the `DoCmd` object.

Referring to objects and collections

Objects in the object model all have a syntax that works like this: You start with the largest, most encompassing object, and work your way down to the most specific object, property, or method. Sort of like a path to a filename, as in `C:\My Documents\MyFile.doc`, where you start with the largest container (disk drive `C:`), down to the next container (the folder named `My Documents`), and then to the specific file (`MyFile.doc`).

For example, the `Application` object refers to the entire Access program. It includes a `CurrentProject` object. If you were to look up the `CurrentProject` object in the Object Browser and view its Help window, you see `CurrentProject` houses several collections, including one named `AllForms`. The `AllForms` collection contains the name of every form in the current database.

The `AllForms` collection, in turn, supports a `Count` property. That property returns the number of forms in the collection. Say that you have a database open and that database contains some forms. If you go to the Immediate window and type

```
? Application.CurrentProject.AllForms.Count
```

and then press Enter, the Immediate window displays a number matching the total number of forms in the database.

At the risk of confusing matters, typing the following line in the Immediate window returns the same result:

```
? CurrentProject.AllForms.Count
```

The shortened version works because the `Application` option is the default parent object used if you don't specify a parent object before `CurrentProject`. (The `Application` object is the parent of `CurrentProject` because `CurrentProject` is a member of the `Application` object library.)

The bottom line is that when you see a bunch of words separated by dots in code (such as `CurrentProject.AllForms.Count`), be aware that those words refer to some object. In a sense, the words are a path to the object — going from the largest object down to a single, specific object, property, method, or event. You can use the Object Browser as a means of looking up the meanings of the words to gain an understanding of how the pre-written code works.

As you gain experience, you can use the Object Browser to look up information about objects, collections, properties, methods, events, and constants within your code. For now, consider the Object Browser as a tool for discovering VBA as you go.

Choosing object libraries

Most likely, the object libraries that appear automatically in the Object Browser's Project/Library drop-down list are all you need. However, should a given project require you to add some other object library, follow these steps to add it:

1. **Choose Tools➪References from the Visual Basic Editor main menu.**

   The References dialog box opens.

2. **Choose any library name from the list.**

   In the unlikely event that you need a library that isn't in the list — but you know you stored it on your hard drive — click the Browse button, navigate to the folder that contains the object library you need, click its name, and then click the Open button.

3. **Click OK when the object libraries you need have check marks.**

   The Project/Library list in the Object Browser now includes all the libraries you selected in the References dialog box.

Closing the Visual Basic Editor

When you're done working in the Visual Basic Editor, you can close it by using whichever of the following techniques is most convenient for you:

- Choose File➪Close and return to Microsoft Access from the Visual Basic Editor main menu.
- Click the Close button in the upper-right corner of the Visual Basic Editor program window.
- Right-click the Visual Basic Editor button on the taskbar, and then choose the Close option from the shortcut menu.
- Press Alt+Q.

Access continues to run even after you close the Visual Basic Editor window.

Discovering Code as You Go

Most beginning programmers start by working with code they pick up elsewhere, such as code generated by code wizards, or code copied from a Web site. You can also create VBA code, without writing it, by converting any macro to VBA code.

Converting macros to VBA code

Any macro you create in Access can be converted to VBA code. Converting macros to code is easier than writing code from scratch. For example, say you need to write some code because a macro can't do the job. But a macro can do 90 percent of the job. If you create the macro and convert it to VBA code, 90 percent of your code is already written. You just have to add the other ten percent (which is especially helpful if you can't type worth beans).

See Book VI, Chapter 1 for how macros work and how to create them.

As an example, suppose you click the Create tab, click the last Other button, and then choose Macros, as in Figure 1-15, to create a new macro.

Then you create your macro. The macro can be as large or as small as you want. Figure 1-16 shows a small simple example of a macro that shows a message on the screen. After you create your macro, close and save it. For this example, say I saved the macro in Figure 1-16 with the name `TinyMacro`.

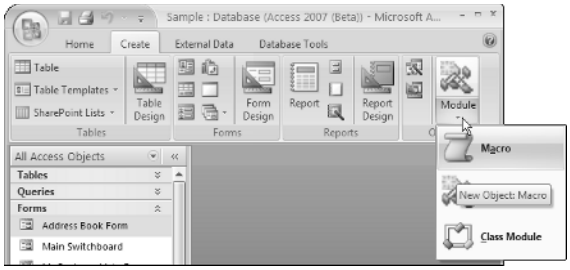

Figure 1-15: Create a new macro.

Figure 1-16: Sample `TinyMacro` macro.

When you convert a macro to VBA code, you actually convert all the macros in the macro group to code. Follow these steps for the basic procedure of converting macros to VBA:

1. **Choose Macros from the top of the Navigation Pane. Or choose All Access Objects and expand the Macros category.**
2. **Click the name of the macro you want to convert.**
3. **Click the Database Tools tab.**
4. **Click the Convert Macros to Visual Basic button shown at the mouse pointer in Figure 1-17.**

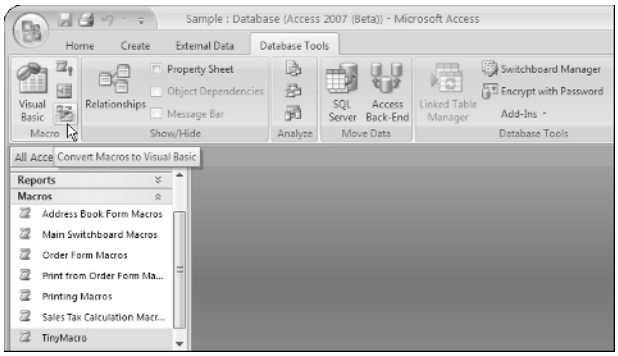

Figure 1-17: Convert Macros to Visual Basic button.

A dialog box appears, asking whether you want to include error-handling code or comments in the code. If you want to keep the code relatively simple, you can clear the first option and select only the second option.

5. **Click the Convert button and then click OK when your conversion is complete.**

To see the name of the converted macro, expand the Modules category as in Figure 1-18. The name of the module is `Converted Macro -` followed by the name of the macro you converted.

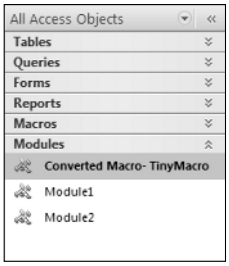

Figure 1-18: `TinyMacro` converted to module named `Converted Macro - Tiny Macro`.

To see the converted macro as VBA code, double-click its name. Like all VBA code, the code from the converted macro opens in the Visual Basic Editor Code window, as shown in Figure 1-19.

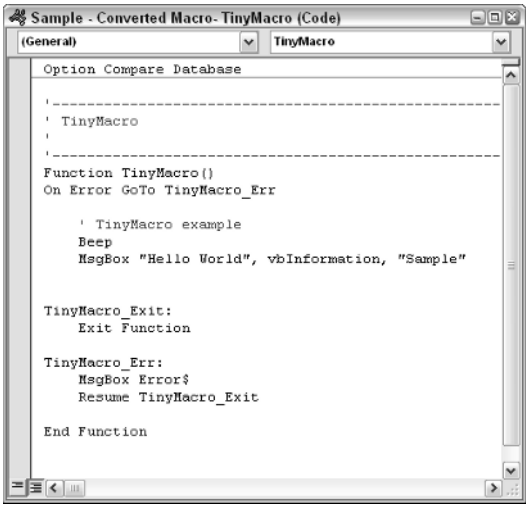

Figure 1-19: Converted macro in Code window.

Cutting and pasting code

Many VBA programmers post examples of code they've written on Web pages. When you come across some sample code you want to incorporate into your own database, retyping it all into the Visual Basic Editor is not necessary. Instead, just use standard Windows cut-and-paste techniques to copy the code from the Web page into the Visual Basic Editor.

Say you come across some code in a Web page you want to use in your own database. Here's the sequence:

1. **In the Web page, you drag the mouse pointer through the code you want to copy to select that code. Then press Ctrl+C to copy that selected code to the Windows Clipboard.**
2. **Back in Access, create a new module or open an existing module in which you want to place the code.**
3. **In the Code window, click at the position where you want to put the copied code. Then paste the code to the cursor position by pressing Ctrl+V.**

Bear in mind, however, that just pasting code into the Code window doesn't make the code do anything. Most code examples are based on a sample database. Just dropping the example into your database may not be enough to get it to work.

When you copy and paste from a Web page, you might get some HTML tags, weird characters, weird spacing, and so forth. If that happens, you can copy the code from the page and paste it into a simple text editor like Notepad first. That should get rid of any unusual tags and characters. Then copy and paste the text from the Notepad document into the VBA Editor's Code window.

But even if you do find an example that's generic enough to work in any database, the code won't actually do anything until some event in your database triggers it into action. We'll look at the many ways you can trigger code into action in the next chapter.